

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Zsolt Kiss

Počítačové a komunikačné siete

Analyzátor sieťovej komunikácie

Zadanie 1

Piatok 8:00-9:50

Ing. Matej Janeba

Obsah

Hlavná myšlienka.....	3
Navrhnuté riešenie	3
Algoritmus programu.....	5
Opis používateľského rozhrania	6
Dôležitá poznámka.....	11
O zdrojovom kóde.....	12
Globálne premenné	12
Vlastné štruktúry	12
Dôležité funkcie	13
Pomocné funkcie.....	16
Zhodnotenie	17

Hlavná myšlienka

Zadanie je vypracované v programovacom jazyku C. Úlohou bolo navrhnutie programového analyzátora Ethernet siete, ktorý analyzuje komunikácie v sieti zaznamenané v .pcap súbore a poskytuje informácie o komunikáciách.

Navrhnuté riešenie

Program na začiatok inicializuje potrebné premenné, otvorí viaceré txt súbory a pcap súbor. Následne načíta hodnotu, ktorú mu zadáme z klávesnice (možnosť menu). V menu sa nachádza aj ukončenie aplikácie (0).

Vyberieme si jednu z možností a program načíta rámce od prvého až po posledný. V mojom programe analyzujem len rámce typu Ethernet II podrobnejšie. Typ rámca zistíme takto, že pozrieme si na 12. a 13. bajt. Ak tie bajty sú väčšie ako 05DC, tak hovoríme o ethertype Ethernet II. Ak nie sú väčšie ako 05DC, tak zistíme podľa nasledujúcich 2 bajtov o aký typ 802.3 protokolu sa jedná.

802.3 RAW by mal nasledujúce hodnoty FF, 802.3 LLC + SNAP by mal nasledujúce hodnoty AA a ak to nie je ani jeden z nich je to 802.3 LLC. Pri 802.3 vypisujeme len jej typ MAC adresy, číslo rámca, dĺžka PCAP api a dĺžka rámca prenášaného po médiu.

Prvé 6 bajtov rámca <0-5> tvoria cieľovú MAC adresu, ďalšie 6 bajty <6-11> tvoria zdrojovú MAC adresu. Nasledujúce 2 bajty hlavičky je Ethertype. Dĺžky rámcov pcap api zistíme pomocou knižnice pcap.h. Dĺžku rámca prenášaného po médiu je o 4 bajtov väčší od dĺžky PCAP API, ale jeho minimálna dĺžka je 64 bajtov

1. Ak typ prenosu je Ethernet II, tak prehl'adávame podrobnejšie
2. Pozeráme na ethertype, ktorý nám hovorí o protokole 3 vrstvy ak je:

0800 – IPv4

- Následne sledujeme protokol (23. bajt), ktorý môže byť:

1	ICMP
2	IGMP
6	TCP
9	IGRP
11	UDP
2F	GRE
32	ESP
33	AH
39	SKIP
58	EIGRP
59	OSPF
73	L2TP

Pri TCP a UDP nachádzajú na 4. vrstve OSI modelu. Pri oboch prípadoch sledujeme zdrojové (34., 35. bajt) a cieľové porty (36., 37. bajt). V prípade ICMP sledujeme typ správy.

TCP:	
7	ECHO
13	CHARGEN
14	FTP DATA
15	FTP CONTROL
16	SSH
17	TELNET
19	SMTP
35	DOMAIN
4F	FINGER
50	HTTP
6E	POP3
6F	SUNRPC
77	NNTP
8B	SMB
8F	IMAP
B3	BGP
185	LDAP
1BB	HTTPS
1BD	MICROSOFT DS
438	SOCKS

UDP:	
7	Echo
13	Chargen
25	Time
35	DNS
43	Bootps DHCP
44	Bootpc DHCP
45	TFTP
89	NBNS
8A	Netbios dgm
A1	SNMP
A2	SNMP trap
1F4	Isakmp
202	syslog
208	RIP
829A	Traceroute
7C1	HSRP
14E9	MDNS

ICMP:	
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect
8	Echo
9	Router Advertisement
a	Router Selection
b	Time Exceeded
c	Parameter Problem
d	Timestamp
e	Timestamp Reply
f	Information Request
10	Information Reply
11	Address Mask Request
12	Address Mask Reply
1E	Traceroute

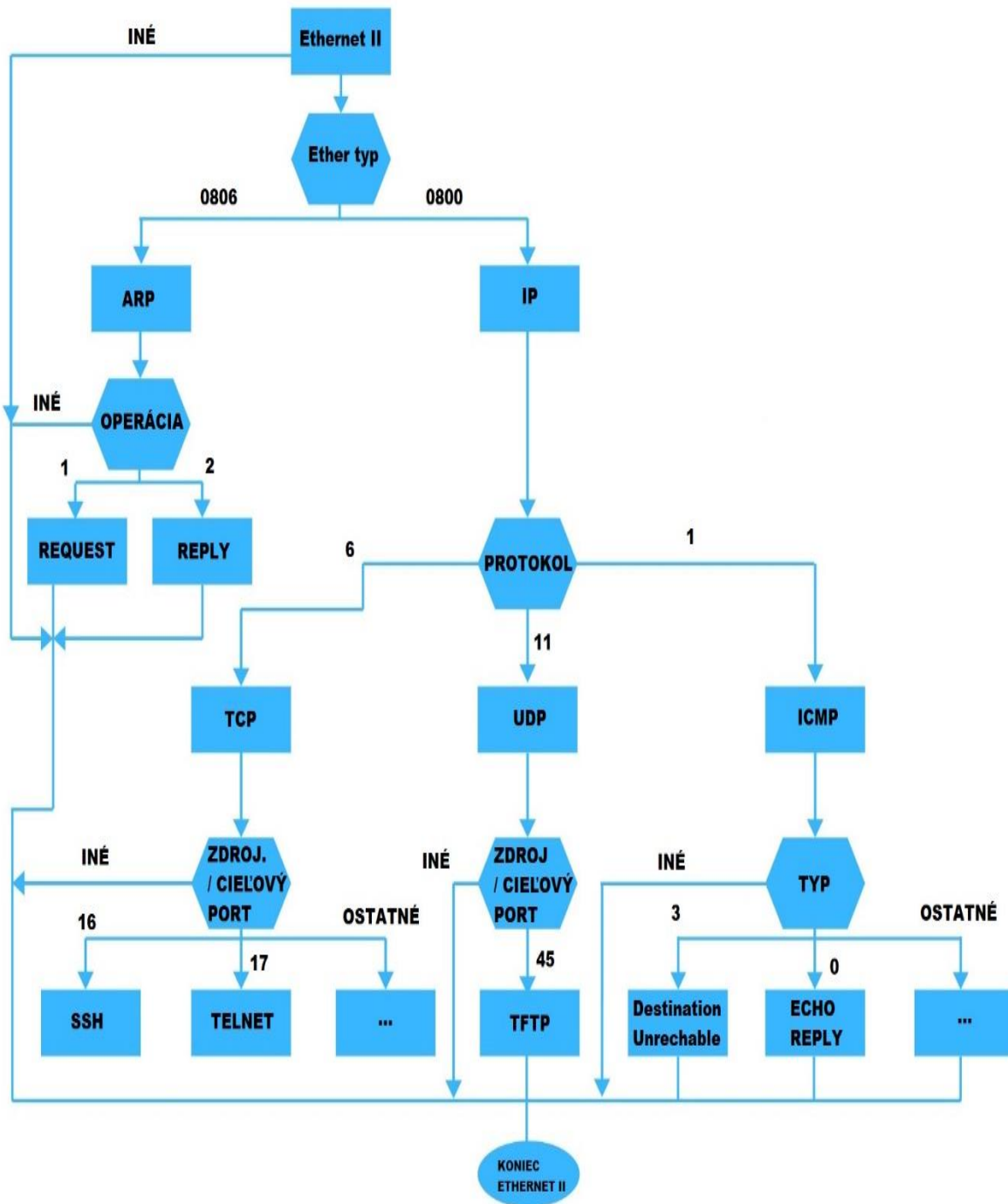
0806 – ARP

- Sledujeme operáciu (21. bajt)

1	Request
2	Reply

Ďalej sledujeme MAC adresu odosielateľa, MAC adresu cieľa (v prípade ak je to Request, tak adresa sú samé 0) IP adresu odosielateľa a IP adresu cieľa.

Algoritmus programu



Opis používateľského rozhrania

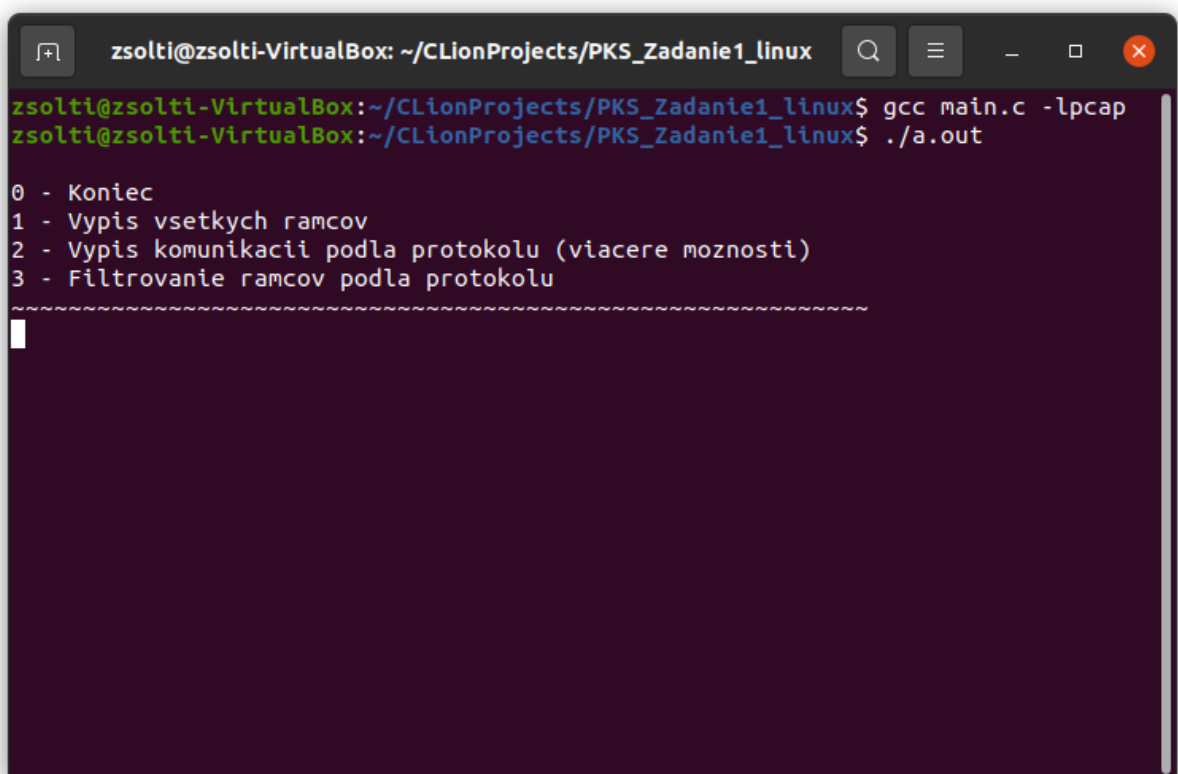
Program som vytvoril tak, aby bol jednoduchý na používanie, preto som implementoval interaktívne menu. Po spustení programu, program nás opýta, aby sme zadali číslo o akú opcii máme záujem. Možnosti sú:

0 – program skončí

1 – program vypíše všetky rámce zo súboru postupne

2 – program vypíše komunikácie podľa bodu 4 a podľa toho, aký protokol mu zadáte. Nie je case sensitive. Možnosti sú: http, https, telnet, ssh, ftp control, ftp data, tftp, icmp, arp.

3 – program vypíše len rámce, podľa toho, aký protokol mu zadáte. Nie je case sensitive. Funguje ako filter pre jednotlivé protokoly.



```
zsolti@zsolti-VirtualBox: ~/CLionProjects/PKS_Zadanie1_linux
zsolti@zsolti-VirtualBox:~/CLionProjects/PKS_Zadanie1_linux$ gcc main.c -lpcap
zsolti@zsolti-VirtualBox:~/CLionProjects/PKS_Zadanie1_linux$ ./a.out

0 - Koniec
1 - Vypis vsetkych ramcov
2 - Vypis komunikaci podla protokolu (viacere moznosti)
3 - Filtrovane ramcov podla protokolu
~~~~~
█
```

Ak používateľ vybral 1, program mu vypíše všetky rámce a ich údaje. Na konci sú uvedené IP adresy všetkých vysielajúcich uzlov, a IP adresa uzla, ktorý odoslal najviac paketov.

```
zsolti@zsolti-VirtualBox: ~/CLionProjects/PKS_Zadanie1_linux

~~~~~
ARP-Request, IP Adresa: 147.175.98.227, MAC Adresa: FF FF FF FF FF FF
Zdrojova IP: 147.175.98.30, Cielova IP: 147.175.98.227
ramec 235
dlzka ramca poskytnuta pcap API - 60 B
dlzka ramca prenasaneho po mediu - 64 B
Ethernet II
ARP
Zdrojova MAC adresa: 00 E0 29 05 51 2B
Cielova MAC adresa: FF FF FF FF FF FF

ff ff ff ff ff ff 00 e0 29 05 51 2b 08 06 00 01
08 00 06 04 00 01 00 e0 29 05 51 2b 93 af 62 1e
00 00 00 00 00 00 93 af 62 e3 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00

~~~~~
ramec 236
dlzka ramca poskytnuta pcap API - 114 B
dlzka ramca prenasaneho po mediu - 118 B
802.3 LLC
IPX (Novell NetWare)

Zdrojova MAC adresa: 00 04 76 13 97 DF
Cielova MAC adresa: FF FF FF FF FF FF

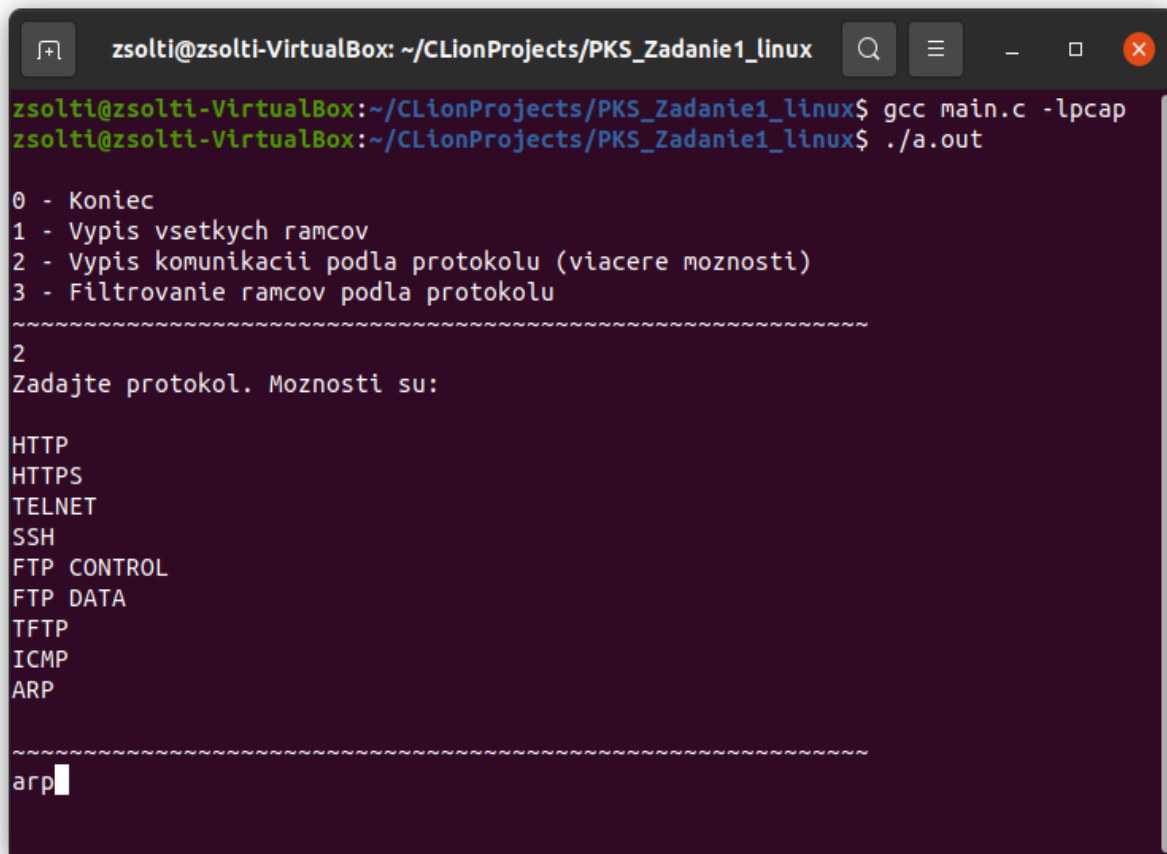
ff ff ff ff ff ff 00 04 76 13 97 df 00 64 e0 e0
03 ff ff 00 60 00 04 30 09 80 00 ff ff ff ff ff
ff 04 52 30 09 80 00 00 04 76 13 97 df 40 10 00
02 06 4e 45 4e 49 47 4d 41 21 21 21 21 21 21 21
21 21 41 35 35 36 39 42 32 30 41 42 45 35 31 31
43 45 39 43 41 34 30 30 30 30 34 43 37 36 32 38
33 32 00 30 09 80 00 00 04 76 13 97 df 40 08 00
01 54

~~~~~
IP adresy vysielajucich uzlov:
147.175.98.238
147.175.98.1
147.175.98.232
147.175.99.30
147.175.98.178
147.175.98.219
147.175.98.28
147.175.98.40
69.56.135.106
147.175.98.230
147.175.98.166
Adresa uzla s najvacsim poctom odoslaných paketov:
69.56.135.106 64 paketov

0 - Koniec
1 - Vypis vsetkych ramcov
2 - Vypis komunikacii podľa protokolu (viacere moznosti)
3 - Filtrovanie ramcov podľa protokolu

~~~~~
```

Ak používateľ vybral 2, program ho požiada, aby zadal aj názov protokolu, ktorý chce, aby mu zobrazili. Možnosti sú HTTP, HTTPS, TELNET, SSH, FTP CONTROL, FTP DATA, TFTP, ICMP, ARP.



```
zsolti@zsolti-VirtualBox: ~/CLionProjects/PKS_Zadanie1_linux
zsolti@zsolti-VirtualBox:~/CLionProjects/PKS_Zadanie1_linux$ gcc main.c -lpcap
zsolti@zsolti-VirtualBox:~/CLionProjects/PKS_Zadanie1_linux$ ./a.out

0 - Koniec
1 - Vypis vsetkych ramcov
2 - Vypis komunikacii podľa protokolu (viacere moznosti)
3 - Filtrovanie ramcov podľa protokolu
~~~~~
2
Zadajte protokol. Moznosti su:

HTTP
HTTPS
TELNET
SSH
FTP CONTROL
FTP DATA
TFTP
ICMP
ARP

~~~~~
arp
```

V tomto prípade používateľ zadal kľúčové slovo arp, program mu zobrazí ARP dvojice, kategorizované do troch tabuliek; ARP dvojice, ARP Requesty bez Reply a ARP Reply bez Request.


```
zsolto@zsolto-VirtualBox: ~/CLionProjects/PKS_Zadanie1_linux
0 - Koniec
1 - Vypis vsetkych ramcov
2 - Vypis komunikacii podla protokolu (viacere moznosti)
3 - Filtrovane ramcov podla protokolu
~~~~~
2
Zadajte protokol. Moznosti su:

HTTP
HTTPS
TELNET
SSH
FTP CONTROL
FTP DATA
TFTP
ICMP
ARP

~~~~~
arp
Komunikacia c.1
~~~~~
ARP-Request, IP Adresa: 192.168.1.1, MAC Adresa: FF FF FF FF FF FF
Zdrojova IP: 192.168.1.33, Cielova IP: 192.168.1.1
ramec 1
dlzka ramca poskytnuta pcap API - 42 B
dlzka ramca prenasaneho po mediu - 64 B
Ethernet II
ARP
Zdrojova MAC adresa: 00 14 38 06 E0 93
Cielova MAC adresa: FF FF FF FF FF FF

ff ff ff ff ff ff 00 14 38 06 e0 93 08 06 00 01
08 00 06 04 00 01 00 14 38 06 e0 93 c0 a8 01 21
00 00 00 00 00 00 c0 a8 01 01

~~~~~
ARP-Reply, IP Adresa: 192.168.1.1, MAC Adresa: 00 02 CF AB A2 4C
Zdrojova IP: 192.168.1.1, Cielova IP: 192.168.1.33
ramec 2
dlzka ramca poskytnuta pcap API - 60 B
dlzka ramca prenasaneho po mediu - 64 B
Ethernet II
ARP
Zdrojova MAC adresa: 00 02 CF AB A2 4C
Cielova MAC adresa: 00 14 38 06 E0 93

00 14 38 06 e0 93 00 02 cf ab a2 4c 08 06 00 01
08 00 06 04 00 02 00 02 cf ab a2 4c c0 a8 01 01
00 14 38 06 e0 93 c0 a8 01 21 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00

~~~~~
ARP Requesty bez Reply:
~~~~~
Subor neobsahoval ARP Requesty bez Reply.
~~~~~
ARP Reply bez Requestu:
~~~~~
Subor neobsahoval ARP Reply bez Requestu.
~~~~~

0 - Koniec
1 - Vypis vsetkych ramcov
2 - Vypis komunikacii podla protokolu (viacere moznosti)
3 - Filtrovane ramcov podla protokolu
~~~~~

```

Ak používateľ vybral 3, program ho požiada, aby zadal aj názov protokolu, ktorý chce, aby mu zobrazili. Bod 3 je odlišný od bodu 2 v tom, že táto možnosť nehľadá komunikácie a spojenia, jednoducho vypíše rámce, ak ich zdrojový alebo cieľový port zhoduje s údajmi v textových súboroch. Napríklad ak používateľ zadá protokol http, je to podobné, keby sme vo Wiresharku spravili vyhľadávanie tcp.port == 80. Na konci výpisu sa zobrazí aj počet nájdených protokolov daného typu. Implementácia tohto funkcionality nebola povinná.

```
zsoliti@zsoliti-VirtualBox: ~/CLionProjects/PKS_Zadanie1_linux
3
Zadajte protokol.
ftp control
ramec 3
dlzka ramca poskytnuta pcap API - 62 B
dlzka ramca prenasaneho po mediu - 66 B
Ethernet II
Zdrojova MAC adresa: 00 14 38 06 E0 93
Cielova MAC adresa: 00 02 CF AB A2 4C
IPv4
zdrojova IP adresa: 192.168.1.33
cielova IP adresa: 193.0.6.140
TCP
FTP CONTROL
zdrojovy port: 3742
cielovy port: 21

00 02 cf ab a2 4c 00 14 38 06 e0 93 08 00 45 00
00 30 14 67 40 00 80 06 5d 0b c0 a8 01 21 c1 00
06 8c 0e 9e 00 15 0e 97 37 4f 00 00 00 00 70 02
ff ff a5 30 00 00 02 04 05 b4 01 01 04 02

ramec 4
dlzka ramca poskytnuta pcap API - 62 B
dlzka ramca prenasaneho po mediu - 66 B
Ethernet II
Zdrojova MAC adresa: 00 02 CF AB A2 4C
Cielova MAC adresa: 00 14 38 06 E0 93
IPv4
zdrojova IP adresa: 193.0.6.140
cielova IP adresa: 192.168.1.33
TCP
```

```
zsolto@zsolto-VirtualBox: ~/CLionProjects/PKS_Zadanie1_linux
zdrojova IP adresa: 192.168.1.33
cielova IP adresa: 193.0.6.140
TCP
FTP CONTROL
zdrojovy port: 3742
cielovy port: 21

00 02 cf ab a2 4c 00 14 38 06 e0 93 08 00 45 00
00 28 14 a0 40 00 80 06 5c da c0 a8 01 21 c1 00
06 8c 0e 9e 00 15 0e 97 37 b7 9a 70 a2 6f 50 10
fe 41 96 5b 00 00

~~~~~
ramec 53
dlzka ramca poskytnuta pcap API - 54 B
dlzka ramca prenasaneho po mediu - 64 B
Ethernet II
Zdrojova MAC adresa: 00 14 38 06 E0 93
Cielova MAC adresa: 00 02 CF AB A2 4C
IPv4
zdrojova IP adresa: 192.168.1.33
cielova IP adresa: 193.0.6.140
TCP
FTP CONTROL
zdrojovy port: 3742
cielovy port: 21

00 02 cf ab a2 4c 00 14 38 06 e0 93 08 00 45 00
00 28 14 a1 40 00 80 06 5c d9 c0 a8 01 21 c1 00
06 8c 0e 9e 00 15 0e 97 37 b7 9a 70 a2 6f 50 14
00 00 94 99 00 00

~~~~~
Tento subor obsahoval 32 protokolov typu ftp control.
```

Dôležitá poznámka

Program bol vytvorený v jazyku C použitím knižnice libpcap na operačnom systéme Linux Ubuntu. Keďže v programe používam niektoré funkcie, ktoré sú exkluzívne pre Linux, program nie je možné bez modifikácie spustiť na operačnom systéme Windows.

V programe si treba zmeniť tie riadky kódu, kde načítame súbor .pcap a textové súbory. Pre textové súbory tieto cesty nastavíme vo funkcii `void openTxtFiles()`.

Pre súbor .pcap, cesta sa nachádza na začiatku `int main()`

O zdrojovom kóde

Globálne premenné

```
bool debugMode = false;
```

Táto premenná, ak je nastavená na true, vypíše dodatočné informácie, ktoré boli užitočné pri debugovaní kódu.

```
bool txtMode = false;
```

Táto premenná, ak je nastavená na true ukončí program po vykonaní výpisu. Je užitočné, ak chceme uložiť výstup vo formáte textového súboru.

Vlastné štruktúry

```
struct IPv4Packet {  
    char* srcIPAddress;  
    int txPackets;  
    struct IPv4Packet* next;  
};
```

Táto štruktúra predstavuje hlavičku pre rámce typu IPv4. V programe som to používal na uloženie IP adres a spočítanie odoslaných pakiet (`int txPackets`) pre jednotlivé IP adresy.

```
struct TCPPacket {  
    int frameNumber;  
    char* srcPort;  
    char* dstPort;  
    char* flag;  
    bool isMarked;  
    struct TCPPacket* next;  
};
```

Táto štruktúra predstavuje hlavičku pre rámce typu TCP. V programe som to používal na overenie otvorenia a ukončenia komunikácií. `char* flag` predstavuje TCP Flag pre daný rámec, ktorý môže mať obsah SYN, ACK, FIN, RST a aji né. `bool isMarked` slúži na to, aby som označil rámce, ktoré už splnili podmienku 3WSH alebo podmienku úspešného ukončenia komunikácie – vtedy táto premenná bude mať hodnotu `true`. Ak sú teda označené, neberieme ich do úvahy pri ďalších hľadaní spojeniach.

```

struct UDPPacket {
    int frameNumber;
    char* srcPort;
    struct UDPPacket* next;
};

```

Táto štruktúra predstavuje hlavičku pre rámce typu UDP. V programe som to používal pri TFTP komunikáciach. Prvý rámec TFTP komunikácie bude mať cieľový port 69. Ak táto podmienka je splnená, pridám rámec do zoznamu, a uložíim jeho zdrojový port. Rámce, ktorý patria do komunikácií budú mať buď cieľový alebo zdrojový port rovnaký ako zdrojový port prvého rámca TFTP komunikácie. Počet TFTP komunikácií v danom súbore je toľko, koľkokrát sa opakuje cieľový port 69.

```

struct ARPPacket {
    int frameNumber;
    char* srcMACAddress;
    char* dstMACAddress;
    char* opCode;
    bool isMarked;
    struct ARPPacket* next;
};

```

Táto štruktúra predstavuje hlavičku pre rámce typu ARP. V programe som to používal pri ARP dvojiciach. **char*** opCode predstavuje Operation code, ktorý bude mať hodnotu Request alebo Reply. **bool** isMarked slúži na to, aby som označil rámce, ktorý sú už spárené– vtedy táto premenná bude mať hodnotu **true**. Ak sú teda označené, neberieme ich do úvahy pri ďalších hľadaní dvojiciach.

Dôležité funkcie

```

void printIPv4PacketList(struct IPv4Packet *temp)

```

Táto funkcia vypíše zoznam vysielajúcich uzlov na konci výpisu 1.

```

void printIPWithTheMostPacketsSent(struct IPv4Packet *start)

```

Funkcia vypíše IP adresu, ktorý odoslal najväčší počet IPv4 rámcov na konci výpisu 1.

```

void printBasicInfo(int frame, int caplen, int len)

```

Funkcia vypíše poradové číslo rámca, dĺžku rámca poskytnutá pcap api v Bajtoch, dĺžku rámca prenášaného po médiu v Bajtoch.

```
void printHexadecimal(int i, const u_char* packet)
```

Funkcia vypíše obsah rámca v hexadecimálnom tvare. Výpis jednotlivých bajtov je tak, že po každom ôsmom bajte nasleduje medzera, a po každom šestnástom bajte nasleduje nový riadok.

```
char* getARPSrcIP (const u_char* packet)
```

Funkcia vráti zdrojovú IP adresu pre rámce typu ARP vo formáte reťazec.

```
char* getARPDstIP (const u_char* packet)
```

Funkcia vráti cieľovú IP adresu pre rámce typu ARP vo formáte reťazec.

```
char* getSrcMAC (const u_char* packet)
```

Funkcia vráti zdrojovú MAC adresu vo formáte reťazec.

```
char* getDstMAC (const u_char* packet)
```

Funkcia vráti cieľovú MAC adresu vo formáte reťazec.

```
char* getSrcIP(const u_char* packet)
```

Funkcia vráti zdrojovú IP adresu pre rámce typu IPv4 vo formáte reťazec.

```
char* getDstIP(const u_char* packet)
```

Funkcia vráti cieľovú IP adresu pre rámce typu IPv4 vo formáte reťazec.

```
char* getSrcPort(const u_char* packet)
```

Funkcia vráti zdrojový port pre rámce typu TCP a UDP vo formáte reťazec.

```
char* getDstPort(const u_char* packet)
```

Funkcia vráti cieľový port pre rámce typu TCP a UDP vo formáte reťazec.

```
char* getTCPFlag(const u_char* packet)
```

Funkcia vráti TCP Flag pre rámce typu TCP vo formáte reťazec.

```
char* getFrameType(const u_char* packet)
```

Funkcia vráti ethertype vo formáte reťazec.

```
char* get<x>FromTXT(const u_char* packet, FILE* <y>)
```

Tieto funkcie majú za úlohu vrátiť reťazec, ktorá je spojená s hodnotou v textovom súbore.

Následne tieto reťazce porovnávame s inými reťazcami, alebo ich len vypíšeme.

```
char* verify3WHS(struct TCPPacket *temp, struct TCPPacket  
*temp2, struct TCPPacket *temp3)
```

Táto funkcia skontroluje otvorenie komunikácie s procesom Three-Way Handshake. Ak nájde takú komunikáciu, tak funkcia vráti reťazec s poradovým číslom prvého rámca (to je rámec, ktorý poslal SYN) a port, na ktorom sa uskutočnilo otvorenie spojenia. Ak takú komunikáciu nenájde, vráti reťazec s obsahom "0 0".

```
char* verifyTermination(struct TCPPacket *temp4, struct  
TCPPacket *temp5, int comStart, const char* clientsSourcePort)
```

Táto funkcia skontroluje úspešné ukončenie komunikácie. Úspešne ukončiť komunikáciu môžeme viacerými spôsobmi;

- Prvý spôsob je ak jedna strana odošle rámec, ktorý má TCP Flag FIN a na to odpovedá druhá strana s rámcom ktorý má TCP Flag tiež FIN.
- Druhý spôsob ukončenia komunikácie je ak jedna strana odošle rámec, ktorý má TCP Flag FIN a na to odpovedá druhá strana s rámcom ktorý má TCP Flag RST.
- Tretí spôsob ukončenia komunikácie je ak jedna alebo druhá strana pošle rámec ktorý má TCP Flag RST. Pred tým nebol žiadny FIN.

Ak nájde takú komunikáciu, tak funkcia vráti reťazec s obsahom portu, na ktorom sa uskutočnilo ukončenie spojenia. Ak takú komunikáciu nenájde, vráti reťazec s obsahom "0".

```
char* connectARPPairs (struct ARPPacket *temp, struct ARPPacket
*temp2)
```

Táto funkcia skontroluje v súbore ARP dvojice. Postupne prechádzame všetky ARP Requesty, a hľadáme taký Reply, ktorá má takú cieľovú MAC adresu, akú zdrojovú MAC adresu mal Request. Ak nájdeme takú dvojicu, tak uložíme ich poradové čísla do jedného reťazca, a funkcia vráti tento reťazec. Ak sme prešli všetky rámce a nenašli sme dvojicu, tak funkcia vráti reťazec s obsahom "00".

Pomocné funkcie

```
bool find<x>PacketInList(struct <x>Packet* head, ...)
```

Tie funkcie slúžia na to, aby som skontroloval, či daný záznam je už prítomný v zozname, alebo nie. Ak už v zozname existuje, viackrát tam nepridám.

```
void insert<x>PacketToList(struct <x>Packet **headRef, ...)
```

Tie funkcie slúžia na vloženie rámcov do spájaného zoznamu.

```
void delete<x>PacketList(struct <x>Packet ** headRef)
```

Tie funkcie slúžia na vymazanie spájaných zoznamov.

```
void print<x>Packet(struct <x>Packet *node)
```

Tie funkcie slúžia na výpis informácií o daného rámca. Používal som ich len pri debugovaní.

```
void printMenu()
```

Výpis hlavného menu. Výstupný txt súbor neobsahuje tento text.

```
void seekToNextLine()
```

Pomocná funkcia pre scanf, aby nebol problém s načítaním čísla v menu.

```
void openTxtFiles(FILE **_802_3SAPs, FILE **_802_3Protocols,
FILE **ethertypes, FILE **IPProtocols, FILE **TCPPorts, FILE
**UDPPorts, FILE **ICMPPorts, FILE **ARPOperations)
```


Funkcia otvorí textové súbory. Ak niektorý zo súboroch nie je možné otvoriť, vypíše chybovú hlášku.

```
int differenceSetOperation(int* excludeFrames, int excludeSize,  
int x, const int* bufferArraySet, int* finalSet)
```

Funkcia spraví množinovú operáciu rozdiel z dvoch množín. Pri ARP dvojiciach ak máme už nejaké dvojice, tak uložíme ich poradové číslo do jednej množiny. Pri výpise ARP Requestov bez Reply a ARP Reply bez Request som použil túto funkciu, aby vynechal tie rámce, ktoré sú už “spárené”.

Zhodnotenie

Počas vytvorenia tohto programu som sa dozvedel veľa nových vecí, čo sa týka sieťovej komunikácie. Program pracuje s dátami optimálne, na uloženie MAC adries a IP adries som používal reťazce. Poradové číslo rámca vo všetkých výpisoch sa zhoduje s číslom rámca v analyzovanom súbore. Vo všetkých výpisoch sú uvedené (ak existujú) použité protokoly, zdrojový a cieľový port, zdrojové aj cieľové adresy. Myslím si, že môj program splní všetky podmienky a je implementované všetko, čo bolo v zadaní uvedené.